Arrays

Chapter 12

One-Dimensional Arrays

- If you wanted to read in 1000 ints and print them in reverse order, it would take a program that's over 3000 lines long.
- There's got to be a better way.
- Don't worry there is.
- An array of size 1000 would do the trick.

Example

```
int main()
{
  int values[1000];
  for ( int i=0; i<1000; i++ )
     cin >> values[i]'
  for ( int i=0; i<1000; i++ )
     cout << values[i];
  return 124;
}
```

Array Declaration

- First give the type of array such as int, double, float, bool, etc.
- Then the name of the array
- Finally, the size of the array inside of square brackets
- Ex-
  - bool TruthArray[12];
  - The size of the array must be a constant int

Information about Arrays

- One dimensional array
  - A structured collection of components all of the same type, that is given a single name. Each component is accessed by an index that indicates the component's position within the collection.
- Array position is always started at 0 and goes up to one less then the size

Accessing Individual Components

- You can access any one particular element if you need or want
  - float angle[4];
  - angle[0] = 1.2;
  - angle[1] = 3.4;
  - angle[2] = 0.0;
  - angle[3] = 45.6;
  - angle[4] does not exist

Things You Can Do with an Array Element

- angle[2] = 9.6;
- cin >> angle[2];
- cout << angle[3];
- y = sqrt(angle[1]);
- x = 6.8 * angle[0] + 7.5;

Things You Shouldn't Do with an Array

- Out-of-bounds array index:
  - An index value that, in C++, is either less than 0 or greater than the array size minus 1.

Initializing Arrays

- You can declare and initialize an array all at one time
- int age[5] = { 23, 10, 16, 37, 12 };
- You can also omit the size of the array and do something like this
- float tempurature[] = { 0.0, 112.37, 98.6 };

Aggregate Array Operations

Operation	Allowed?
I/O	No
Assignment	No
Arithmetic	No
Comparison	No
Argument Passage	Reference only
Return from a function	No

Aggregate Operations

- You can write functions that handle all of those operations for you
- For example:

```
void CopyArray( const int x[], int y[], int size )
{
   for (int i=0; i<size; i++)
     y[i] = x[i];
}
```

Software Engineering

- It's always better to declare a const int as the size for the array
- const int BUILDING_SIZE = 350;
- int occupants[BUILDING_SIZE];
- You can use BUILDING_SIZE in your for loops as the stop condition
- Now if you need to change the size of your build you change it in one place
- You can make this a global constant and so all scopes will have access to the constant

Arrays and Functions

- You can pass arrays as arguments to functions
- You will do this for your last two projects
- Arrays are pass-by-reference by default.
- You cannot get pass-by-value
- You can pass-by-const-reference by putting const before the variable type in the definition and prototype.

Arrays and Functions

- Function invocation
- CopyArray(MyArray, MyOtherArray, 10);

- Function definition

```
void CopyArray( const int x[], int y[], int size )
{
   for (int i=0; i<size; i++)
     y[i] = x[i];
}
```

Array Elements and Functions

- It is possible to pass just one location of an array to a function.
- This is pass-by-value by default
- It can also be pass-by-reference and pass-by-const-reference just like every other variable

Example

Function invocation

```
Swap( MyArray[4], MyArray[132] );
```

Function

```
void Swap( int &x, int &y )
{
   int temp = x;
   x = y;
   y = temp;
}
```

Helpful Idea

- When thinking about arrays, when ever you type just the name of the array without any brackets you mean the entire array
- When you type the array name with brackets and the number inside, you mean just that particular location

Two-Dimensional Arrays

- Two dimensional arrays are the same in use except you need an extra set of brackets to indicate the second dimension
- Example
  - const int NUM_ROWS = 100;
  - const int NUM_COLS = 9;
  - float alpha[NUM_ROWS][NUM_COLS];
- In C++, typically the rows come first.

Processing 2-D Arrays

- Assuming the array declaration from the previous slide, we can:
- Initialize the array

```
for ( int row=0; row<NUM_ROWS; row++ )
  for ( int col=0; col<NUM_COLS; col++ )
    alpha[row][col] = 0.0;
```

More Processing

- Sum Columns

```
for ( int col=0; col<NUM_COLS; col++ )
{
  total = 0;
  for ( int row=0; row<NUM_ROWS; row++ )
    total += alpha[row][col];
  cout << "Column sum: " << total << '\n';
}
```

2-D Arrays and Function

- To pass a 2-D array into a function you need to specify the size of the second dimension in the prototype and function definition
- For example

```
void Copy( int X[][MAX_COL],
           const int Y[][MAX_COL],
           int MAX_ROW);
```

Activity

- Write a function that takes two arrays of ints, of size Size, and swaps them.
- You can use this prototype if you wish:

```
void SwapArray( int ArrayOne[], int ArrayTwo[],
  int Size );
```